# Lesson 4
# Understanding Web Applications

## Learning Objectives

Students will learn about:

- Web Page Development
- ASP.NET Application Development
- IIS Web Hosting
- Web Services Development

## OBJECTIVE1: Understanding Web Page Development

A Web page is a document that is served over the World Wide Web (WWW) and can be displayed by a Web browser. Web pages are developed using the Hypertext Markup Language (HTML) and are stored on a Web server. Web browsers download the requested HTML from the Web server and render it on the user's screen

The World Wide Web (also known as WWW or "the Web") is a system of interconnected hypertext documents and other resources (such as images and video) that can be accessed via the Internet. Multiple technologies work together to make the WWW possible. In this section, we will discuss two of these technologies

### Hypertext Transfer Protocol (HTTP)

### Hypertext Markup Language (HTML)

HTTP is the underlying communication protocol used by the World Wide Web. HTTP provides the common language that Web servers and Web browsers use in order to communicate.

HTTP uses a Uniform Resource Locator (URL) to uniquely identify and address each resource on the Internet. A URL is essentially a Web address and looks like this: http://www.microsoft.com/en/us/default.aspx. Each URL starts with a protocol. In this example, the protocol is HTTP. You may also notice the HTTPS (secure HTTP) protocol in use for secure applications in which data needs to be encrypted before it is transmitted over the network. After the protocol, the next part of a URL is the address of the Web server (here, www.microsoft.com), followed by the location of the resource within the Web server (/en/us/), and finally, the requested resource itself (default.aspx). All documents, images, videos, and other resources on the Web are identified by a URL.

When a browser sends an HTTP request for a Web page to a Web server (both the Web page and the server are identified by a URL), the server prepares an HTTP response for the browser. This response specifies the content and layout of the Web page.

***TakeNote***

The terms "Internet" and "the Web" are often used interchangeably, but they are actually distinct and should

not be confused. The Internet is a global data communications system that provides connectivity among computers. In contrast, the Web is one of several services available on the Internet that allows users to access hyperlinked resources.

The language that the Web server and Web browser use to describe a Web page is Hypertext Markup Language (HTML). HTML is a text-based language that uses various markup tags that describe how content is displayed. HTML allows images, videos, and other objects to be referenced in a file to create multimedia Web pages. HTML can also embed scripts (such as JavaScript) that affect the behavior of Web pages, and it can be used to include cascading style sheets (CSS) to define the formatting and layout of a page's content. The Web browser reads the HTML code and renders the results on the screen.

A Web page may contain hyperlinks to other resources, such as images and videos. Each of these resources is identified by its own URL. Thus, in order to render a page completely, the browser will make additional HTTP requests to get these resources and display them as part of the Web page.

In the following sections, you'll learn more about the various components that make up a Web page, including HTML, CSS, and JavaScript

## Understanding HTML

Hypertext Markup Language (HTML) is the language used by Web servers and browsers to describe a Web page

The purpose of HTML is to provide a standard language for describing Web pages so that different Web browsers can understand this language and display the corresponding page. HTML is a text-based language, which means that you can write and edit HTML pages using any text editor. When HTML is sent to a Web browser, the complete text of the page is sent. In fact, most browsers allow you to view the HTML source code for a Web page

HTML consists of a set of tags (also called as HTML elements) that define the structure and content of a page. For example, the <html> tag specifies the beginning of an HTML document. HTML tags are always surrounded by angle brackets and always used in pairs. In particular, each starting tag has a matching ending tag. Ending tags contain a forward slash to indicate that they are such. For example, the ending tag for <html> is </html>.

An HTML page has two distinct parts: a header and a body. The header is enclosed within the <head> and </head> tags and is used to provide a document title and links to external items that may be used in the page, such as CSS files and JavaScript files. The body is enclosed within the <body> and </body> tags, and it is used to provide the complete structure and content of the page that will be displayed within a Web browser

Here is an example of an HTML tag that displays an image:

```
<img  height="400px"  width="400px" alt="Mimas Cassini" src=
"http://upload.wikimedia.org/wikipedia/commons/b/bc/Mimas_Cassini.jpg"/>
```

Notice that the <img> tag specifies additional attributes. For example, the src attribute specifies the location of the image file, and the height and the width attributes specify what dimensions to use when rendering the image in a browser.

Now, consider another example of an HTML tag:
<a href="http://en.wikipedia.org/wiki/Mimas_(moon)"> Saturn's moon</a>

Here, <a> is the anchor tag, which is used to create hyperlinks on a Web page. The href attribute associated with this tag specifies the target URL, and the text within the anchor tag is that which is displayed as a link.

***TakeNote***

This lesson does not cover all HTML elements. To learn more about these elements, search for "HTML elements" on the MSDN
HTML is text-based when it comes to writing the code. HTML provides tags to embed pictures, audio, video, and many other types of multimedia and interactive content on a Web page

The following exercise demonstrates the steps involved in creating an HTML document

**Illustration1: WORK WITH HTML**

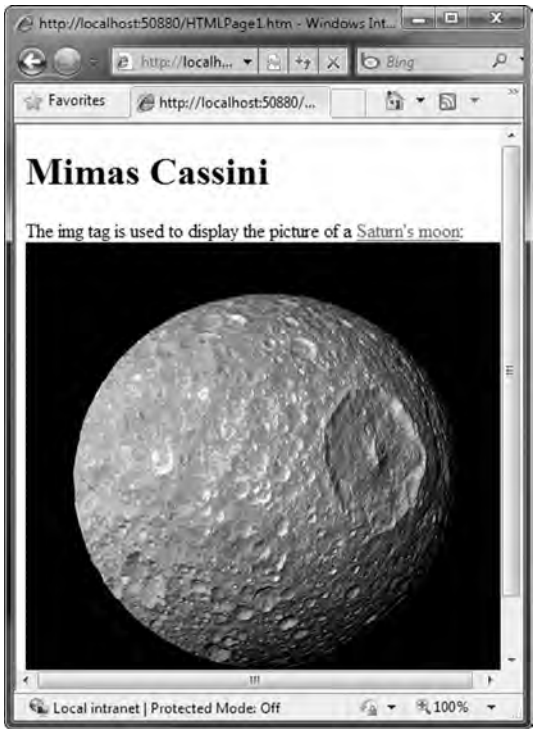
GET READY. To create an HTML document, perform these actions:

1. Open Visual Studio. Create a new project based on the ASP.NET Empty Web Application template.
2. Name the project WorkingWithHTML and name the solution Lesson04.
3. Select Project > Add New Item, then select the HTML Page template. Name the file default.htm.
4. Replace the default code in the HTML file with the following:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Saturn's Moon</title>
</head>
<body>
<h1>Mimas Cassini</h1>
The img tag is used to display the picture of a
<a href="http://en.wikipedia.org/wiki/Mimas_(moon)"> Saturn's moon</a>: <br />
<img height="400px" width="400px" alt="Mimas Cassini" src="http://goo.gl/3BeK"/>
</body>
</html>
```

5. Select Debug > Start Debugging (or press F5). The default.htm page will open in a Web browser. The output should look similar to Figure 4-1, where you can see the <img> and <a> tags in action.

Figure4-1


http://localhost:50880/HTMLPage1.htm - Windows Int...
http://localh...
Bing
Favorites
http://localhost:50880/...
Mimas Cassini
The img tag is used to display the picture of a Saturn's moon:
Local intranet | Protected Mode: Off
100%

## Understanding Cascading Style Sheets

Cascading style sheets (CSS) enable you to store a Web page's style and formatting information separate from the HTML code. This separation makes it easier to update the look and feel of your Web site. Visual Studio includes tools to build and preview your style sheets.

CSS is a language that describes information about displaying a Web page. When rendering Web pages in a browser, HTML specifies what will be displayed, and the cascading style sheets (CSS) specify how that material will be displayed. For example, HTML can specify that your document has a H1 heading with a given text, and CSS can specify the font and color that will be applied to that heading.

CSS enables you to separate the layout of a Web page from its content. This separation allows you to change one without affecting the other. Mixing content and style together reduces the maintainability of a Web site. For example, say you want to change the color and font of all H1 headings on your Web site. One approach may be to open an HTML editor and modify each file on the Web site that uses the H1 tag. This might be an acceptable solution if the Web site has just one or two pages, but what if the site has a large number of pages—for example, 50 or 100? Imagine manually changing each page! If such changes are requested often, the Web development process will be boring and error prone. After all, how can you ensure that you did not miss any H1 tags?

Fortunately, with CSS, you can put all such styling information in a separate file and connect that file to all pages on a Web site. Then, once the CSS file is set up, you can modify any style (such as the color and font of H1 headings) simply by changing the style in the CSS file—and this single change will affect all pages on the Web site.

***TakeNote***

When used effectively, CSS is a great tool for increasing site-wide consistency and maintainability

## DESIGNING CASCADING STYLE SHEETS

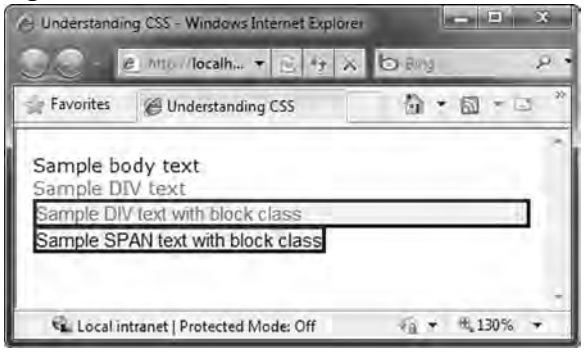
The CSS language is text-based and easy to read and understand. The following is an example of an HTML page that defines CSS styles:

```
<html  xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Understanding  CSS</title>
<style  type="text/css">
body
{
font-family:  Verdana;
 font-size: 9pt;
}
div
{
color:Red
```

```
}
block
{
background-color: Yellow;
 border-color: Blue;
border-width:   thin;
border-style:  outset; font-family:   Arial
}
</style>
</head>
<body>
Sample body text <br />
<div>Sample DIV text</div>
<div class="block">Sample DIV text with block class</div>
<span class="block">Sample SPAN text with block class</span>
</body>
</html>
```

Note that the CSS definitions must be inside the <style> element and are defined under the <head> element. This particular CSS defines two element styles and a class style. The first style applies to the HTML body element and specifies that all text in the body element should use the Verdana font with 9-point font size. The second element style specifies that the text within the DIV element should be written in red. Finally, a class named "block" is defined. CSS class definitions are prefixed with a dot (".") . The contents of any HTML element that uses this class will be displayed with yellow background and a border. When you display this particular page in a browser, it should appear as shown in Figure 4-2.

Figure 4-2




In the figure, notice that the highlighted text is displayed as a result of the block class. However, the block CSS class does not specify the color of the text. In the first line of high- lighted text, the block class is applied to the DIV element; in the second line of highlighted text, the block class is applied to the SPAN element. In the first case, because the block class is applied to a DIV text, the color style of the DIV element is carried over in the final rendering.

In the previous example, the CSS file was written inside the HTML file. A more useful approach is to write the CSS in its own separate file and then link the HTML file to this CSS file. You will learn how to do so in

the following exercise.

**Illustration2: WORK WITH CSS FILES**

GET READY. To write a CSS file and link it to an HTML file, perform these steps:

1. Add a new project based on the ASP.NET Empty Web Application template to the Lesson04 solution. Name the project UnderstandingCSS.
2. Select Project > Add New Item. Select the Style Sheet template. Name the file styles.css. Replace the default code in the file with the following

```
body
{
font-family:  Verdana; font-size: 9pt;
}
div
{
color:Red
}
block
{
background-color: Yellow;
border-color: Blue;
 border-width:   thin;
border-style:  outset;
font-family:   Arial;
}
```

3. Select Project > Add New Item, then select the HTML Page template. Name the file default.htm. Replace the default code in the file with the following:

```
<!DOCTYPE html PUBLIC
"–//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1–
transitional.dtd">
<html  xmlns="http://www.w3.org/1999/xhtml">
<head>
<link  rel="STYLESHEET"      type="text/css"  href="styles.css"  />
<title>Understanding  CSS</title>
</head>
<body>

Sample body text <br />
<div>Sample DIV text</div>
<div class="block">Sample DIV text with block class</div>
<span class="block">Sample SPAN text with block class</span>
```

```
</body>
</html>
```

4. Select Debug > Start Debugging (or press F5). The default.htm page will open in a Web browser, and the output should be similar to Figure 4-2 (presented earlier).

As shown in the example exercise, the HTML <link> element is used to link a CSS file with an HTML page:

```
<link  rel="STYLESHEET"
type="text/css" href="styles.css" />
```

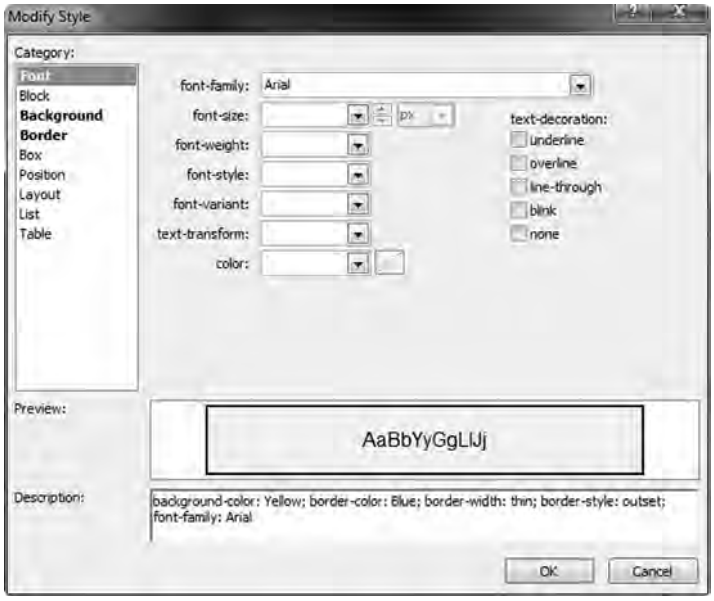
The <link> element is always put within the <head> element, and the href attribute specifies the address of the CSS file to use. To link multiple pages with the same CSS file, you'll need to put the <link> element within each HTML page.

Visual Studio includes a built-in style designer that can help you design new CSS styles or modify existing styles. When you open a .css file, you'll see a new menu named Styles. From this menu, you can create a new style by selecting Styles > Add Style Rule. You can also modify the currently selected style by selecting the Styles > Build Style option. This option opens the Modify Style dialog box, as shown in Figure 4-3

***TakeNote***

When CSS are stored in separate files, the user's browser will download and store these files locally. Therefore, they can be used on multiple pages without any need to download them again. This reduces unnecessary data transfer

Figure 4-3 The Modify Style Dialog box




## Understanding JavaScript

JavaScript is a client-side scripting language that runs inside Web browsers to help create far more interactive Web pages than are possible with only HTML. JavaScript is used in millions of Web pages and is supported by all modern Web browsers.

JavaScript is used to make Web sites more responsive and Web pages more interactive. JavaScript accomplishes this by executing the code on the client side (the Web browser) and by minimizing unnecessary round-trips to and from the Web server. Let's take an example in which a user needs to enter personal details on a Web page, such as name, email address, and phone number. A common requirement is to perform data validation to ensure that the input fields are not empty and the user's email address and phone number have been provided in the required form. Without JavaScript, you would need to submit the form to the server, which will perform data validation and return the results to the client. This trans- mission of information takes time and degrades the user experience. However, a JavaScript solution can perform this type of data validation from within the browser, providing a better user experience.

As previously mentioned, JavaScript code executes locally within the Web browser (the client), as opposed to on the Web server. Therefore, JavaScript is sometimes also called a client-side scripting language, and programming with JavaScript is referred to as client-side programming.

The runtime behavior of client-side code execution depends on the browser that executes it. However, this behavior is independent of server technology or programming framework. Thus, for JavaScript that is being executed in a Web browser, it does not matter whether the Web page was generated by ASP.NET or PHP or whether the page is being served by a Windows Web server or a Linux Web server.

JavaScript and the C# programming language both use a syntax influenced by the C programming language. Still, JavaScript and C# are very different in how they are executed. In particular, JavaScript is executed by the Web browser, and JavaScript code is interpreted rather than compiled, as in the case of C#.

All JavaScript code must be placed inside the <script> element. The <script> element is usually placed inside the <head> element, although this is not required. Multiple <script> elements may exist within a single page. To see JavaScript in action, try the following exercise

***TakeNote***

Many modern Web sites provide a highly interactive experience rivaling that of desktop applications. Such sites can be developed using Ajax programming. Ajax is shorthand for "Asynchronous JavaScript and XML." Ajax uses JavaScript extensively in order to provide responsive Web applications. The ASP.NET AJAX framework lets you implement Ajax functionality on ASP.NET Web pages

**Illustration3: WORK WITH JAVASCRIPT**

GET READY. To begin working with JavaScript, perform the following tasks:

1. Add a new project based on the ASP.NET Empty Web Application template to the Lesson04 solution. Name the project UnderstandingJavaScript.
2. Select Project > Add New Item, then select the HTML Page template. Name the file default.htm. Replace the default code in the file with the following:

```
<!DOCTYPE html PUBLIC
"–//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1–
transitional.dtd">
<html  xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Understanding JavaScript</title>
<script type="text/javascript" language="javascript">
username = prompt("Please enter your name");
message = "Hello, " + username +
". Your name has ";
nameLen = username.length; if (nameLen > 5)
message = message + "more than ";
 else if (nameLen < 5)
message = message + “less than” ;
else
message = message + “exactly”;
message = message +”5characters”;
alert(message)
</script>
</head>
<body>
```

```
</body>
</html>
```

3. Select Debug > Start Debugging (or press F5). The default.htm page will open in a Web browser. Notice that the JavaScript code prompts you to enter your name. Once you have done so, a dialog box displays a message based on the length of your name, as shown in Figures 4-4 and 4-5.

Figure 4-4 Java Script user Prompt

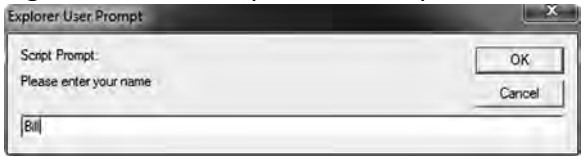
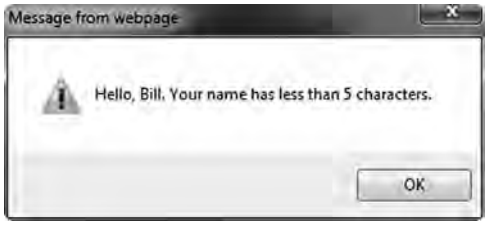



Figure4-5 JavaScript dialog box




As with CSS files, you can also put your JavaScript code in a separate file and link this file with an HTML file by using the script element, as shown below:

```
<script src="SampleScript.js">
</script>
```

Here, the SampleScript.js file contains all the JavaScript code, and the script element links to this file by using the src attribute. Storing JavaScript in external files offers many advantages:

- Improved maintainability: If you use the same JavaScript code on each page of a Web site, you can store the code on a central page rather than repeating it on every page. This way, when it's time to modify the JavaScript code, you'll have to change the code in only one place.
- Improved performance: Storing JavaScript code in a separate file reduces the size of a Web page. Also, browsers can download and cache the external JavaScript file so that it is not downloaded again unless it is modified.

Visual Studio includes full IntelliSense support for JavaScript code. Even ASP.NET controls, like the TreeView control or the validation controls, use JavaScript where possible to render content dynamically

***TakeNote***

Although all modern browsers support JavaScript, they can be set so that JavaScript is turned off. You can use a <noscript> element to display a specific message to users who opt not to run JavaScript

## Understanding Client-Side vs. Server-Side Programming

Whether a program is client-side or server-side depends on where the program is ultimately executed.

Client-side programming refers to programs that execute completely on a user's local computer. Examples of client-side programs include the Windows Forms application and JavaScript code that executes within a Web browser. Client-side programs do not consume server resources.

On the other hand, server-side programming refers to programs that are executed completely on a server and make use of the server's computational resources. Here, the only client resources that are used are those involved in actually retrieving the processing results from the server. Web applications and Web services are examples of server-side programming. This type of programming uses a server-side technology such as ASP.NET, PHP, or Ruby on Rails.

Recently, hybrid applications that use both client- and server-side programming have become increasingly popular. For instance, you can design smart-client applications that run locally on client computers but make use of Web services to accomplish certain tasks. In fact, Ajax applications use a mix of server-side programming and client-side code to create interactive and highly responsive Web applications

ASP.NET allows you to create applications that completely execute on the server or hybrid Ajax applications that provide fast, responsive interfaces while storing most of their data on the Web

## OBJECTIVE2: Understanding ASP.NET Application Development

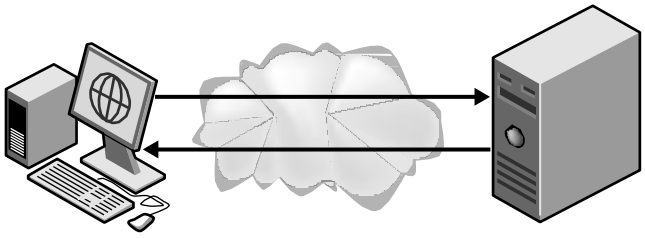
ASP.NET is the part of the .NET Framework that enables you to develop programmable Web forms and Web services. As with any .NET Framework application, you can develop ASP.NET applications in any language that is compatible with the .NET common language runtime, including Visual Basic and C#.

The ASP.NET infrastructure has two main parts:
A set of classes and interfaces that enables communication between the Web browser and Web server. These classes are organized in the System.Web namespace.
A runtime process, also known as the ASP.NET worker process (aspnet_wp.exe), that handles the Web request for ASP.NET resources.

At a higher level, an ASP.NET Web application is executed through a series of HTTP requests and response messages between the client browser and the Web server. This process occurs as follows:

1. The user requests a resource from a Web server by typing a URL in the Web browser.
2. The browser sends an HTTP request to the destination Web server.
3. The Web server analyzes the HTTP request and searches for a process capable of executing the request.
4. The result of the HTTP request is returned to the client browser in the form of an HTTP response.
5. The browser reads the HTTP response and renders it as a Web page to the user.The process is represented in figure 4-6

Figure4-6



As a developer, you should know what happens behind the scenes when a Web server executes a request for an ASP.NET page. The following steps describe this process:

When the Internet Information Service (IIS) receives an HTTP request, it uses the file- name extension to determine which Internet Server Application Programming Interface (ISAPI) program to run to process the request. When the request is for an ASP.NET page, it passes the request to the ISAPI DLL capable of handling requests for ASP.NET pages, which is aspnet_isapi.dll. The aspnet_isapi.dll process passes the request to the ASP.NET worker process (aspnet_ wp.exe), which fulfills the request.The ASP.NET worker process compiles the .aspx file into an assembly and instructs the Common Language Runtime (CLR) to execute the assembly.

When the assembly executes, it takes the services of various classes in the .NET Framework class library to accomplish its work and generate response messages for the requesting client.The ASP.NET worker process collects the responses generated by the execution of the Web page, creates a response packet, and passes it to the aspnet_isapi.dll process.Aspnet_isapi.dll forwards the response packet to IIS, which in turn passes the response to the requesting client machine.

Prior to execution, each ASP.NET page is converted into a class. This class derives most of its functionality from the System.Web.UI.Page class. The Page class provides several important properties, such as Request, Response, Session, and Server.

***TakeNote***

The Page class provides several important methods and properties that control how a page request is processed. For the complete list of methods and properties, visit http://msdn. microsoft.com/en-us/library/system.web.ui.page.aspx.

## Understanding ASP.NET Page Life Cycle and Event Model

During its execution, an ASP.NET page passes through many distinct stages of processing. Each of these stages itself goes through specific processing steps, such as initialization, loading, running event-handler code, and rendering.

As a page executes, it goes through various stages of processing. The page also fires a few events to which you can attach an event handler in order to execute custom code at different stages of page processing. In fact, ASP.NET developers must have a good understanding of the page life cycle so that they can write code that is executed at exactly the desired stage of page processing.

***TakeNote***

In a typical contact form, you enter information and press a submit button. When you submit this page, the page

can process the submitted data to take some action, such as storing the information in a database or sending an email. In many cases, the initial page is displayed again with a confirmation of the form submission.
A postback occurs when the information is posted to the same Web page for processing. A postback is different from the initial load of the page because the page receives additional information (such as form data) as part of the postback.

Table 4.1 lists the different life cycle stages and their associated events.

| | | |
|---|---|---|
| Page request | When a request for a page is received, the page life cycle begins. At this point, ASP.NET decides whether the page can be readily served from the cache or whether it needs to be parsed and compiled | |
| Start | The start stage involves determining whether the request is a postback or a new request. Several page properties, such as Request, Response, IsPostBack, and UICulture, are set at this stage | PreInit |
| nitialization | During the initialization stage, all the controls on the page are initialized and made available. An event handler for the Init event is the best place for code that you want to be executed prior to further page processing. | Init |
| Load | If the request is a postback, the load stage is used to restore control properties with information from view state and control state. A method that handles the Load event is the best place to store initialization code for any controls specific to this page | Load |
| Postback event handling | If the request is a postback, the control event handlers are called during this stage. Then, the input | |

| | | |
|---|---|---|
| | values are validated and the IsValid property for the Page class is set. | |
| Prerendering | This stage signals that the page is just about to render its contents. An event handler for the PreRender event is the last chance to modify the page's output before it is sent to the client | PreRender |
| Rendering | At this stage, the page calls the Render method for each control and populates the response that will be sent to the Web browser. | |
| Unload | During the unload stage, the response is sent to the client and page cleanup is performed. As part of this cleanup, page properties such as Request and Response are discarded | Unload |

When you want to handle an event, you must write code that registers a method to handle the event (also called as event handler) with the event. This is usually done using the common event registration pattern employed throughout the .NET Framework:

object.event += new EventHandler(eventhandler);

Here, replace object with the name of the object that exposes the event, event with the name of the event, and eventhandler with the name of the method that handles the event.

Note, however, that ASP.NET provides six special methods that are recognized as event han- dlers by default and do not need the above registration code. These are the specially named methods Page_Init, Page_Load, Page_DataBind, Page_PreRender, and Page_Unload. These methods are treated as event handlers for the corresponding events exposed by the Page class. This automatic event wiring is controlled by the AutoEventWireup attribute of the @Page directive. By default, the value of this attribute is true, meaning that these specially named methods are automatically wired up with their corresponding events

**Illustration4: UNDERSTAND THE ASP .NET PAGE LIFE CYCLE**

GET READY. To see how different events of the Page class are executed, perform the following actions:

1. Create a new project based on the ASP.NET Empty Web Application template to the Lesson04 solution. Name the project PageEvents.

2. Select Project > Add New Item. Select the Web Form template. Name the file WebForm1.aspx.

3. In the HTML markup for the page (WebForm1.aspx), make sure that the AutoEventWireup attribute for the @Page directive is set to true:

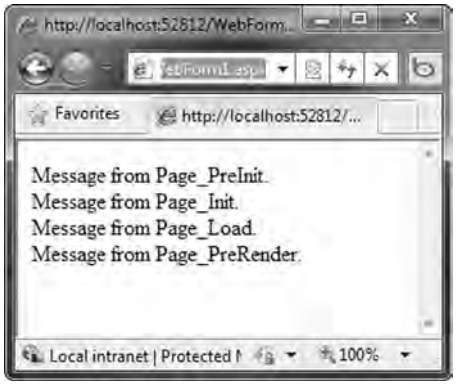
   ```
   <%@ Page Language="C#" AutoEventWireup="true"
   CodeBehind="WebForm1.aspx.cs"
   Inherits="PageEvents.WebForm1" %>
   ```

4. Right-click in the code window and select View Code from the shortcut menu to switch to the code view. Replace the code in the code-behind file (WebForm1.aspx.cs) with the following

```
using System;
namespace PageEvents
{
public partial class WebForm1
: System.Web.UI.Page
{
protected void Page_Load (object sender, EventArgs e)
{
Response.Write
("Message from Page_Load. <br/>");
}
protected void Page_Init (object sender, EventArgs e)
{
Response.Write
("Message from Page_Init. <br/>");
}
protected void Page_PreRender (object sender, EventArgs e)
{
Response.Write
("Message from Page_PreRender. <br/>");
}
protected void Page_PreInit (object sender, EventArgs e)
{
Response.Write
("Message from Page_PreInit. <br/>");
}
}
}
```

5. Select Debug > Start Debugging (or press F5). The default.htm page will open in a Web browser, as shown in Figure 4-7

Figure4-7




In the example exercise, note that the characters <% and %> are used to embed code blocks in the HTML markup of a page. The code inside these embedded blocks is executed during the page's rendering stage. Within the embedded code blocks, the syntax <%=expression> is used to resolve an expression and return its value into the block. For example, consider the following block of code:

<% = DateTime.Now.ToShortDateString() %>

When executed, this code will display the current date in italicized format:
12/01/2010

The @Page directive specifies various attributes that control how ASP.NET will render a page. For example, in this exercise, attributes of the @Page directive specify the following:

- C# is the programming language for this Web page (Language="C#")
- The page's events are auto-wired (AutoEventWireup=true)
- The name of the code file that contains the class associated with the page (CodeBehind="WebForm1.aspx.cs")
- The class name for the page to inherit (Inherits="PageEvents.WebForm1")

## Understanding State Management

State management is an important issue for Web applications because of the disconnected nature of HTTP. There are both client-side and server-side techniques available for state management

State management is the process of maintaining state for a Web page across round-trips. The values of the variables and controls collectively make up the state of a Web page.

ASP.NET provides several techniques for preserving state information across page postbacks. These techniques can be broadly categorized as either client-side or server-side, depending on where the resources are consumed

## INTRODUCING CLIENT-SIDE STATE MANAGEMENT

Client-side state management use HTML code and the capabilities of the Web browser to store state information on the client computer. The following techniques are used for storing state information on the client side:

- **Query strings:** Here, state is maintained by putting the data as a set of key-value pairs in the query string portion of a page URL. For example, the following URL embeds a query string key (q) and value (television) pair: http://www.bing.com/ search?q=television. To retrieve the value of the key in an ASP.NET page, use the expression Request.QueryString[“q”]. QueryString is a property of the Request object, and it gets the collection of all the query-string variables.

- **Cookies:** Cookies are small packets of information that are stored by a Web browser locally on the user’s computer. Cookies are commonly employed to store user preferences and shopping cart contents and to give users a personalized browsing experience on subsequent visits to a Web page. The HttpCookie class represents a cookie in your code. The following code shows how to set a cookie on a client computer:

```
HttpCookie cookie =new HttpCookie("Name", "Bob");

cookie.Expires = DateTime.Now.AddMinutes(10); Response.Cookies.Add(cookie);

Similarly, the following piece of code shows how to read a cookie:
if (Request.Cookies["Name"] != null)
{
name = Request.Cookies["Name"].Value;
}
```

- **Hidden fields**: Hidden fields contain information that is not displayed on a Web page but is still part of the page’s HTML code. Hidden fields can be created by using the <input type=“hidden”> HTML element. The ASP.NET HTML Server control HtmlInputHidden also maps to this HTML element.
- **ViewState:** ViewState is the mechanism ASP.NET uses to maintain the state of controls across page postbacks. To facilitate this, when ASP.NET executes a page, it collects the values of all nonpostback controls that are modified in the code and formats them into

a single encoded string. This string is stored in a hidden field in a control named VIEWSTATE. By default, ViewState is enabled in an ASP.NET application. You can disable ViewState at the level of a control by setting the EnableViewState property of the control to false:

```
<asp:GridView   ID="GridView1"
runat="server" EnableViewState="false" />
```

In addition, you can disable ViewState at the page level by setting the EnableViewState attribute of the Page directive to false:

```
<%@ Page EnableViewState="false" %>
```

Finally, you can disable ViewState at the application level by adding the following line to the web.config file:

```
<pages enableViewState="false" />
```

## INTRODUCING SERVER-SIDE STATE MANAGEMENT

Server-side state management uses server resources to store state information. Using server- side techniques eliminates the possibility that a user will try to hack the client-side code or read the session data. However, storing and processing session information on a server increases the server's load and requires additional server resources to serve the Web pages.

ASP.NET supports server-side state management at two levels:

- **Session state**: An ASP.NET application creates a unique session for each user who sends a request to the application. ASP.NET distinctly identifies each of these sessions by sending a unique SessionId to the requesting URL. This SessionId is transmitted as a cookie or embedded in the URL, depending on the application's configuration. The ability to uniquely identify and relate requests can be used to store session-specific data that is also known as the session state of the Web application. A common example of session state is storing shopping cart contents for users as they browse through a Web- based store.

***TakeNote***

The session state can be configured for storage on another server or a SQL server. This is useful when a user's request can be processed by one of the many servers in a Web farm. . A Web farm is a collection of Web servers used collectively to serve a Web site. Web farms are necessary to support traffic on popular Web sites

- **Application state**: Application state is used to store data that is used throughout an application. Application state can be easily accessed through the Application property of the Page class. This

property provides access to the HttpApplicationState object that stores the application state as a collection of key-value pairs.

The following exercise shows how to use the session state. This exercise involves two Web forms. The WebForm1.aspx gets a user name and stores it in the session state. The form then transfers the user to the WebForm2.aspx, which retrieves the user name from the session.

**llustration5: USE SESSION STATE**

GET READY. To use session state, perform the following steps:

1. Add a new project based on the ASP.NET Empty Web Application template to the Lesson04 solution. Name the project UsingSessionState.
2. Select Project > Add New Item. Select the Web Form template. Name the file WebForm1.aspx
3. Change the HTML markup of the WebForm1.aspx to the following:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="WebForm1.aspx.cs" Inherits="UsingSessionState.WebForm1"    %>
<html  xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:Label ID="Label1" runat="server" Text="Please enter your name:" /><br />
<asp:TextBox ID="TextBox1" runat="server" />
<br /><br />
<asp:Button ID="Button1" runat="server" Text="Submit" onclick="Button1_Click" />
</div>
</form>
</body>
</html>
```

4. Right-click in the code window and select View Code from the shortcut menu to switch to the code-behind file (WebForm1.aspx.cs). Replace the code in that file with the following:

```
using System;
namespace  UsingSessionState
{
public partial class WebForm1
```

```
: System.Web.UI.Page
{
protected void Page_Load (object sender, EventArgs e)
{
if (Session["Name"] != null) Response.Redirect("WebForm2.aspx");
}

protected void Button1_Click (object sender, EventArgs e)
{
Session.Add("Name", TextBox1.Text);
Response.Redirect("WebForm2.aspx");
}
}
}
```

5. Add a new Web Form (WebForm2.aspx) to the project. Change the markup of the page to the following

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="WebForm2.aspx.cs" Inherits="UsingSessionState.WebForm2"    %>
<html  xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:Label ID="Label1" runat="server" /> <br />
<asp:Button ID="Button1" runat="server" Text="Clear Session"
onclick="Button1_Click" />
</div>
</form>
</body>
</html>
```

6. Change the code in the code-behind file (WebForm2.aspx.cs) for the form so that it reads as follows:

```
using System;
namespace  UsingSessionState
{
public partial class WebForm2
: System.Web.UI.Page
{
protected void Page_Load( object sender, EventArgs e)
```

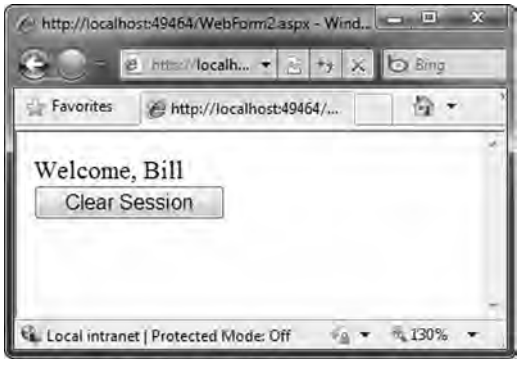
```
{
if (Session["Name"] != null)
Label1.Text = String.Format(
"Welcome, {0}", Session["Name"]);
else
Response.Redirect("WebForm1.aspx");
}
protected void Button1_Click( object sender, EventArgs e)
{
Session.Remove("Name");
Response.Redirect("WebForm1.aspx");
}
}
}
```

7. Select Debug > Start Debugging (or press F5). The WebForm1.aspx page will open in a Web browser, as shown in Figure 4-8. Enter a name and click the Submit button. This page stores the entered name in the session state.

Figure4-8


Next, you'll be transferred to WebForm2.aspx, as shown in Figure 4-9. WebForm2.aspx retrieves the user name from the session state. In the same browser window (so that you are within the same session), try accessing WebForm1.aspx. Notice that as long as the session contains an entry for the name, you will be redirected to WebForm2.aspx. Press the Clear Session button. This clears the session and transfers you to WebForm1.aspx

Figure4-9




## OBJECTIVE3: Understanding IIS Web Hosting

Web hosting involves setting up a Web server with correct code files and settings so that remote users can successfully access a Web application

ASP.NET applications must be deployed on an Internet Information Services (IIS) Web server. IIS is an integral part of Windows Server operating systems and provides functionality for hosting Web sites.

Deploying an ASP.NET application is uncomplicated because ASP.NET provides xcopy deployment. What this means is that all you need to do to deploy an ASP.NET Web site to a Web server is copy the files to the correct locations. You can copy these files using either the Windows xcopy command or a File Transfer Protocol (FTP) application.

***TakeNote***

| Some complex Web applications may require you to deploy DLL files to the global assembly cache (GAC). In such situations, you may actually need to create a Windows Installer package for deployment rather than using xcopy or FTP |
|---|

### Understanding Internet Information Services

Internet Information Services (IIS) is a Web server for hosting Web applications on the Windows operating system. An IIS server uses the concepts of sites, applications, and virtual directories.

You can use IIS to host multiple Web sites and share information with users over the Internet or over an intranet. IIS uses a hierarchical relationship among sites, applications, and virtual directories as a basic

building block for hosting online content.

IIS can be administered using the IIS Manager tool, which is part of the Windows operating system. The IIS Manager tool, as shown in Figure 4-10, provides a graphical user interface to configure Web sites, applications, and virtual directories. The screenshot in Figure 4-10 is from a computer running Windows 7. The user interface of IIS Manager is different on Windows XP.

Figure4-10


## Creating Virtual Directories and Web Sites

A Web site is a container of applications and virtual directories. A virtual directory is an alias that maps to a physical directory on the Web server.

A Web site is a container of applications and virtual directories that can be accessed using a Web address. For example, the URL www.northwind.com may point to a Web site that has many virtual directories, such as orders and accounts, each of which can be accessed in combination with the Web site address—for example, via www.northwind.com/orders and www.northwind.com/account.

A Web server never exposes the actual physical address and location of files to the external world. Instead, it uses a system of aliases that map to the physical directories. These aliases are also called virtual directories. The virtual directories become a part of the URL, as demonstrated in the previous

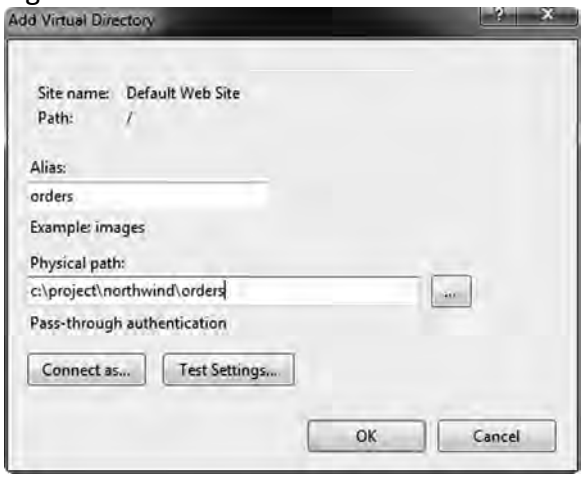
example. When IIS receives a request for such a URL, it maps the virtual directory to the physical location of the files. The following exercise shows how to create a virtual directory

**Illustration 6: CREATE VIRTUAL DIRECTORY**

GET READY. To create a virtual directory using the IIS manager, take the following steps:

1. Open the IIS Manager. To open IIS Manager in Windows 7, type IIS in the Start menu, then click on the Internet Information Service (IIS) Manager shortcut. To access IIS Manager in Windows XP, go to Start > Run, type "inetmgr", and click the OK button.
2. Expand the nodes on the left panel (see Figure 4-10) and select the Default Web Site node.
3. Right-click on the Default Web Site node and select the Add Virtual Directory option from the shortcut menu. In Windows XP, the command will be New > Virtual Directory. At this point, a Virtual Directory Creation Wizard will appear on your screen.
4. In the Add Virtual Directory dialog box, provide an alias and physical path, as shown in Figure 4-11, then click OK.

Figure 4-11




## Deploying Web Applications

Deployment of simple Web sites is accomplished by copying the files to the correct location. To install a complex Web site, you may need to use Windows Installer

There are two primary ways in which you can deploy files to a Web site:

- **Using xcopy or FTP**: Many Web applications and Web services simply require the files to be copied onto the Web server. These sites and services don't require any special actions, such as restarting IIS services, registering the components to Windows Registry, and so on. The xcopy or FTP deployment is ideal in such scenarios.

- **Using Windows Installer**: Windows Installer can perform a number of custom actions during the deployment process. Therefore, it can be used for deploying complex Web sites that require automatically creating virtual directories, restarting services, registering components, and so on.

## OBJECTIVE4: Understanding Web Services Development

A Web service is a software component that can be accessed over a network using standard network protocols such as HTTP. Web services are described using the Web services description language (WSDL).

Web services provide a way to interact with programming objects located on remote computers. What makes Web services special is that all communication between Web service servers and their clients occurs via Extensible Markup Language (XML) messages transmitted over the Hypertext Transfer Protocol (HTTP).

By using these standard technologies, remote objects can be published and consumed by otherwise noncompatible systems. For example, a remote object written in C# and published as a Web service on a Windows Web server can be processed by Java code running on a Linux machine.

Before we get into the details of creating and consuming Web services, let's familiarize ourselves with two key technologies that make Web services possible:

- **Simple Object Access Protocol (SOAP)**
- **Web Services Description Language (WSDL)**

### Introducing SOAP

SOAP is the protocol for exchanging structured information in a Web service communication between two remote computers.

SOAP is the protocol that defines how remote computers exchange messages as part of a Web service communication. SOAP relies on XML as its message format and uses HTTP for message transmission. Using SOAP to communicate has two major benefits. First, because Web service messages are formatted as XML, they're easier for noncompatible systems to understand. Second, because these messages are transmitted over the pervasive HTTP, they can normally reach any machine on the Internet without being blocked by firewalls.

Here's a typical SOAP packet sent from a client to a Web service

```
<?xml version="1.0"  encoding="utf-8"?>
<soap:Envelope
xmlns:xsi=http://www.w3.org/2001/XMLSchema–instance
xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<ToLower   xmlns="http://northwindtraders.com">
<inputString>SAMPLE   STRING</inputString>
</ToLower>
</soap:Body>
</soap:Envelope>
```

As you look at the example, notice some of the obvious elements of this SOAP packet:

- The packet consists of an envelope that contains a body; each is identified with a specific XML tag.
- The body consists of the name of the method to be invoked. In this SOAP packet, the method name is ToLower, and it takes a single parameter by the name of inputString and a given value.

Here is the response packet from the server:

```
<?xml version="1.0"  encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<ToLowerResponse
 xmlns="http://northwindtraders.com">
<ToLowerResult>sample    string</ToLowerResult>
</ToLowerResponse>
</soap:Body>
</soap:Envelope>
```

In the response packet, the ToLowerResponse XML element is the result of the method invocation on the server

## Introducing WSDL

WSDL is an XML-based language for describing Web services.

WSDL stands for ***Web services description language***, and it provides a standard by which a Web service can tell its client what kind of messages it will accept and what results will be returned. A WSDL file acts as the public interface of a Web service and includes the following information:

- The data types it can process
- The methods it exposes
- The URLs through which those methods can be accessed
-

***TakeNote***

| |
|---|
| A Web service can exist without a WSDL file, but you must know the exact incoming SOAP message that the Web service expects before you can use that service. |

## Creating Web Services

In this section, you learn how to create and publish a Web service.

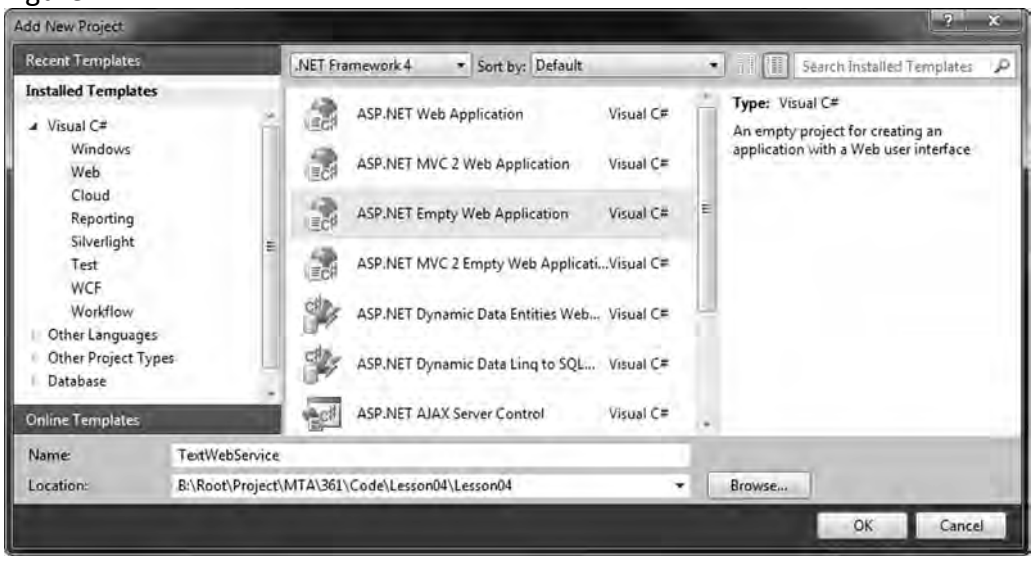
In this section, you'll learn how to create a simple Web service called TextWebService that exposes two methods, ToLower and ToUpper. These methods convert a given string to lower- case and upper-case letters, respectively. Although this example is simple, it covers all aspects of creating a Web service that may involve more complex processing logic

**Illustration7: CREATE A WEB SERVICE**

GET READY. To create a Web service, perform the following actions:

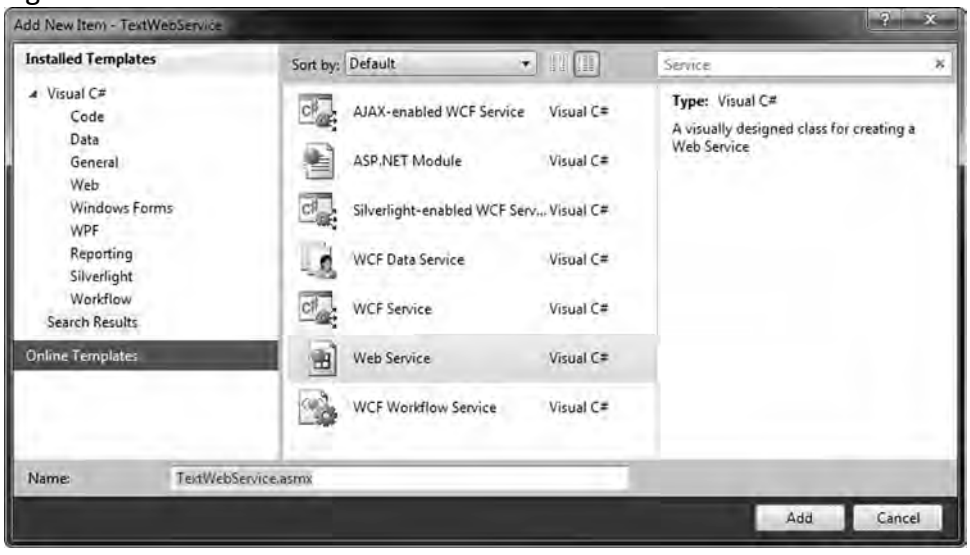
1. Add a new project based on the ASP.NET Empty Web Application template to the Lesson04 solution. Name the project TextWebService, as shown in Figure 4-12.

Figure4-12



2. Select Project > Add New Item from the shortcut menu. Select the Web Service template, as shown in Figure 4-13. Name the Web service TextWebService.asmx.

Figure 4-13




3. Change the default code for the TextWebService class in the TextWebService.asmx.cs file as shown below:

```
[WebService(Namespace = "http://northwindtraders.com/")]
[WebServiceBinding(ConformsTo
= WsiProfiles.BasicProfile1_1)] public class TextWebService:
System.Web.Services.WebService
{
[WebMethod]
public string ToUpper(string inputString)
{
return  inputString.ToUpper();
}
[WebMethod]
public string ToLower(string inputString)
{
return  inputString.ToLower();
}
}
```

4. Select Debug > Build TextWebService to compile the project and ensure that there are no errors. The Web service is now ready for use

In the above code, there are few important things to note. First, notice that each class that is exposed as an XML Web service needs to have a WebService attribute. The WebService attri- bute has a Namespace property that defaults to https://tempuri.org/. Although it is okay to have this value at development

time, the namespace value should be changed before the Web service is published. In fact, each individual Web service must have a unique namespace in order for client applications to distinguish it from other Web services.

Each method that is exposed from the Web service also needs to have a WebMethod attribute. The methods marked with WebMethod attributes are also known as Web methods. The two methods used in this exercise convert a given string to upper-case and lower-case letters, respectively.

To test a simple Web service such as the TextWebService created above, all you need is a Web browser. You can select methods to invoke, pass parameters, and review the return values from within the browser, as shown in the following exercise.

***TakeNote***

| |
|---|
| You can use your company's domain name as part of the namespace to distinguish your Web services from the services published by other companies |

**Illustration7: TEST A WEB SERVICE**

GET READY. To test a Web service, perform the following tasks:

1. Open the TextWebService project that you created in the previous exercise. Select Debug > Start Debugging. A browser will be launched displaying the Web service test page, as shown in Figure 4-14.

Figure4-14

2. On the test page, click on the Service Description link. In this way, you will be able to view the WSDL for this Web service. Click on the Back button to return to the test page.
3. Notice that all the Web methods appear as links on the test page. To invoke a partic- ular Web method, click on its link. After doing so, you should see a page for testing the selected Web method, as shown in Figure 4-15

Figure 4-15

4. Each Web method test page shows the SOAP and other messages that the Web service understands. It also contains a form that allows you to enter method parameters and test the Web method. Enter a test input string and click the Invoke button. You should see a result on the next page, as shown in Figure 4-16

Figure 4-16


Test both methods. When you are finished, close the Web browser.

As demonstrated in the exercise, when you click on the Invoke button, the test page constructs the appropriate SOAP packets for the Web service to process and then displays the results returned from the Web service.

## Consuming Web Services

In this section, you learn how to access Web services from a client application.

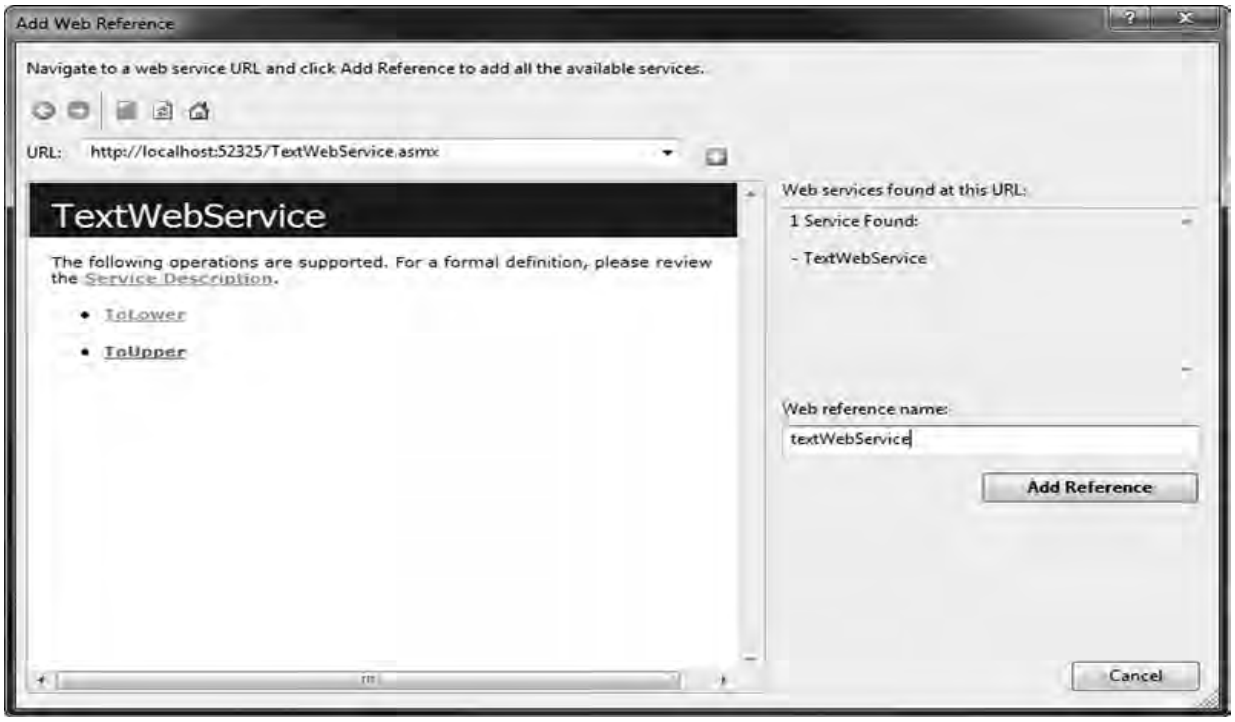
Earlier in the lesson, you learned how to invoke a Web service from the Web service test page. In this section, you'll learn how to call a Web service programmatically from within an ASP.NET client application

**Illustration7: ACCESS A WEB SERVICE FROM A CLIENT APPLICATION**

GET READY. To access a Web service from a client application, perform the following steps:

1. Add a new project to the solution Lesson04 based on the ASP.NET Web Application template. Name the project TextWebServiceClient.
2. Right-click the project's name in the Solution Explorer window, then select the Add Web Reference option from the shortcut menu. In the Add Web Reference dialog box, enter the URL of the service created in the previous exercise, and press Enter. (You can also copy the URL from the browser's address bar.) This action loads the list of operations available on the Web service, as shown in Figure 4-17.

Figure 4-17




3. In the Add Web Reference dialog box, change the name of the Web reference to textWebService and click the Add Reference button. This will add a Web Reference to the project, as shown in Figure 4-18.

Figure4-18


4. Change the code of the Default.aspx to the following:

```
<%@ Page Title="Home Page" Language="C#"
 AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="TextWebServiceClient._Default" %>
<html>
```

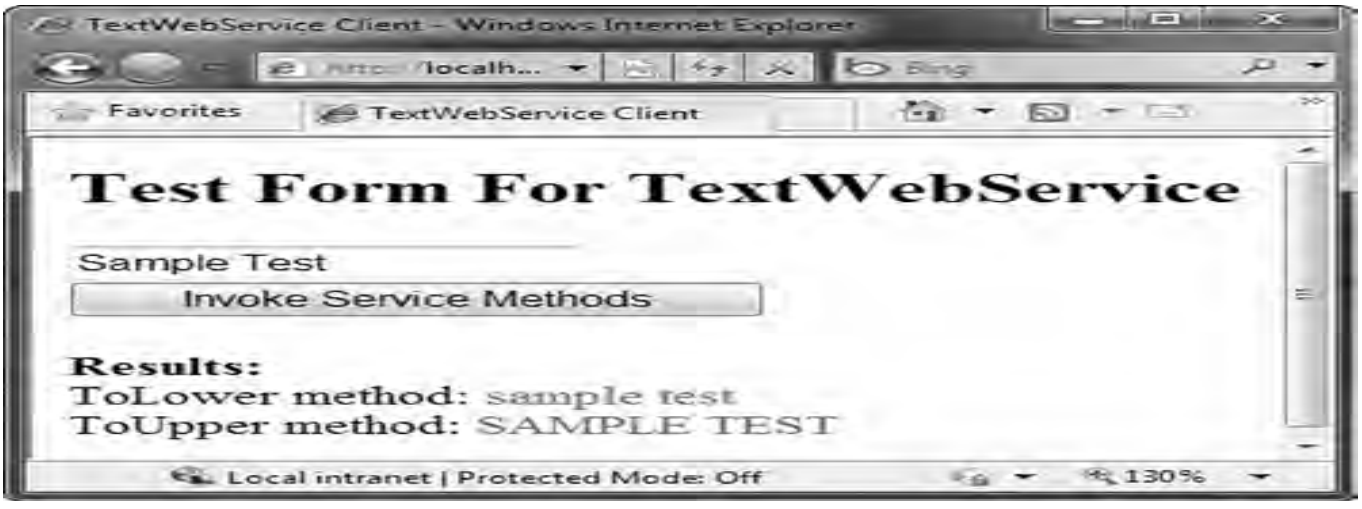
```
<head><title>TextWebService   Client
</title>
</head>
<body>
<form runat="server">
<h2>Test Form For TextWebService</h2>
<p>
<asp:TextBox ID="TextBox1" runat="server" Text="enter text" />
<br />
<asp:Button ID="Button1" runat="server"
Text="Invoke Service Methods"
  onclick="Button1_Click" />
</p>
<p>
<strong>Results:</strong><br  /> ToLower method:
<asp:Label ID="toLowerLabel"\
runat="server"
Text="Label" ForeColor="Green" />
<br />
ToUpper method:
<asp:Label ID="toUpperLabel" runat="server"
Text="Label" ForeColor="Green" />
</p>
</form>
</body>
</html>
```

5. Open the Design view for Default.aspx and double-click the Button control. This adds code for the Click event handler. Modify the code as shown below:

```
protected void Button1_Click( object sender, EventArgs e)
{
var webService =
new  textWebService.TextWebService();
toLowerLabel.Text  =webService.ToLower(TextBox1.Text);
toUpperLabel.Text  = webService.ToUpper(TextBox1.Text);
}
```

6. Select Debug > Start Debugging to run the Web application. Enter some sample text, then click the Invoke Service Methods button. You should see the results from the TextWebService, as shown in Figure 4-19.

Figure 4-19




***TakeNote***

When you invoke a Web service method, you have a choice of using a synchronous or an asynchronous method. You may want to use the asynchronous method to increase the responsiveness of the client application.

In the above exercise, when you add a Web reference, Visual Studio creates a local proxy that represents the remote service. The proxy simplifies communication with the Web service by accepting messages, forwarding them to the Web service, and returning results from the Web service.

You can easily use this proxy to create objects from the Web service and invoke methods. As a result, working with remote objects is similar to working with local objects

When you create a Web reference, Visual Studio reads the appropriate WSDL file to determine which classes and methods are available on the remote server. When you call a method on a remote object, the .NET Framework translates your call and results into SOAP messages and transmits them with no intervention on your part.